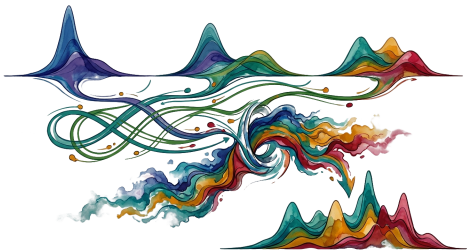
Large State Spaces: Function Approximation

How dynamic programming survives when the value is fitted rather than tabulated

Aurélien Garivier

CR15 Reinforcement Learning — M2, ENS de Lyon

Companion to Chapter 13 of the course notes



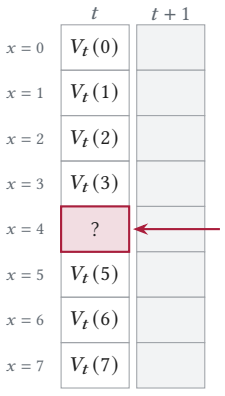
Why tables fail

So far a value function was a *table*: one number per state. Fine for a bike store with 16 stock levels or a small grid; hopeless when the state is

- the **wear of a machine**, a real number: a grid of step h has $1/h$ cells, and d such variables need $(1/h)^d$ cells;
- the **cart-pole** of session H06: position, speed, angle, angular speed, a point of $\mathbb{R}^4$;
- an **Atari screen** (Mnih et al., 2015): 84×84 pixels with 256 gray levels each.

In such problems most states are *never visited twice*.

What does a table know about a state it has never seen? Nothing. Each entry is learned on its own; nothing is passed on to the neighbors.



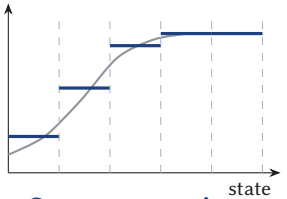
a table of values of the wear, step 1, one column per time

$x = 4$ never visited: the table says nothing, although $V_t(3)$ and $V_t(5)$ are known

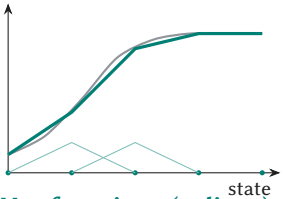
and the wear is a real number what about $x = 3.72$?

The one idea: represent, then generalize

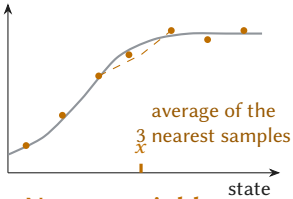
Describe the value function by *a few numbers*, and let the values of unseen states be **inferred** from those of seen ones. The same gray curve, represented with five numbers in three ways:



State aggregation
one number per cell:
a staircase



Hat functions (splines)
one number per knot:
a broken line



Nearest neighbors
the value at x is an average
of sampled values near x

Generalization is the only way out, and it is the whole subject of this lecture: what happens to dynamic programming when every "store $V(s)$" becomes "fit V".

Three families of classes, one fitting procedure

Linear
$V_\theta(s) = \varphi(s)^\top \theta$: fixed features $\varphi(s) \in \mathbb{R}^d$ (cells, hat functions, Fourier), d weights to learn. Session H05: $d = 10$ hat functions of the wear.

Averagers
k-nearest neighbors, kernel smoothing, trees: the value at s is a *weighted average* of sampled values. They never extrapolate.

Neural networks
$Q_\theta(s, a)$ is the output of a network trained by stochastic gradient descent. Flexible; nonlinear, nonconvex training.

Whatever the class, fitting means **regression**: inputs $x_1, \dots, x_n$ drawn from a *design law* μ (*where we look*), targets $y_1, \dots, y_n$ (*what the function should be there*), and

$$f \in \mathcal{F} \text{ approximately minimizing } \sum_{i \le n} \big(f(x_i) - y_i\big)^2.$$

- A regression is good *on average where it looked*: it controls $\|V\|_\mu = \big(\sum_s \mu(s) V(s)^2\big)^{1/2}$.
- Dynamic programming wants the error *everywhere*: its guarantees are about $\|V\|_\infty = \max_s |V(s)|$.

Reconciling "good on average where I looked" with "good everywhere" is the central difficulty.

What is the learner given? Four regimes

The regimes of Chapter 12

(i) Known model — the law of the wear increments, the demand law of the store. The backup $(T^*V)(s)$ is computed *exactly* at any state; we choose where to fit (μ). *Approximate dynamic programming proper: a planning problem.*

(ii) Simulator — a physics engine, a game emulator. For any (s, a), *one* sampled next state and reward: the backup becomes a noisy estimate; μ is still our choice.

(iii) Batch — logs of past decisions (a retailer's history, medical records). Transitions (S_i, A_i, R_i, S_i') collected by someone else; μ is *their* law: *offline* RL.

(iv) Online — a robot learning while it acts: one trajectory, incremental updates, the sampling law is the trajectory's own.

Same regression, different targets, different errors. Known model: exact targets, only the class can be too small. Simulator: noisy targets, an estimation error. Batch: moreover, the data decide which states are seen. Online: all of that, along one trajectory.

The map of the lecture

A simplified version of Table 13.1

Method	Regime	What is approximate	What is guaranteed
Approximate value / policy iteration (Part 1)	(i)	the representation of V; backups exact	errors do not explode: $\times \frac{1}{1-\gamma}$ on the value, $\times \frac{2\gamma}{(1-\gamma)^2}$ on the policy
State aggregation, averagers (Part 1)	(i)	idem, with a fit that never overshoots	convergence; error $\leq$ variation inside a cell $\times \frac{1}{1-\gamma}$
Lookahead, rollout (Part 1)	(i), (ii) at decision time	only the terminal value	k steps ahead multiply the error by γ^{k-1}
Fitted value iteration (Part 2)	(ii)	backups *sampled*; μ chosen	as above, plus an estimation term
Fitted Q-iteration from a batch (Part 2)	(iii)	backups sampled; μ imposed	idem, if the data cover what matters
Linear TD, LSTD (Part 3)	(iv), (iii)	the fixed point of a *projected* equation	convergent on-policy; can diverge off-policy
Linear MDPs, complete classes (Part 4)	any	nothing: the class is closed under the backup	exact
Deep Q-networks (Part 5)	(iv)	everything	no theorem; the structure of Part 1

Three errors keep coming back: **approximation** (class too small), **estimation** (too few samples), **optimization** (solver did not finish). The known-model regime has only the first.

Part 1

Planning with a known model

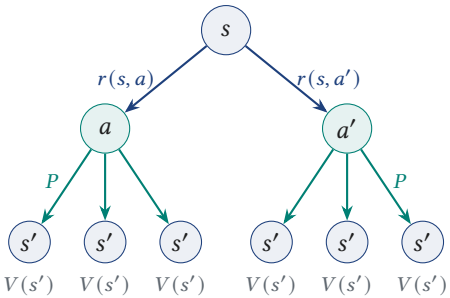
1. Planning with a known model
2. Without the model: fitted value iteration
3. Policy evaluation with linear features: TD and its limits
4. When is approximation benign? Classes closed under the backup
5. Deep Q-networks

Value iteration, recalled (1/2): the backup

One **backup** at a state s is *one step of lookahead*: for each action, the immediate reward plus the discounted average value of the next states; then the best action:

$$(T^*V)(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s' \mid s, a)\, V(s') \right].$$

- It needs the *model* (r and P) and the values of the successors.
- The **greedy policy** of V plays, at s, the action achieving the max.
- The **Bellman residual** $(T^*V)(s) - V(s)$ says how much V moves under one backup: $V^\star$ is the function that does not move at all.



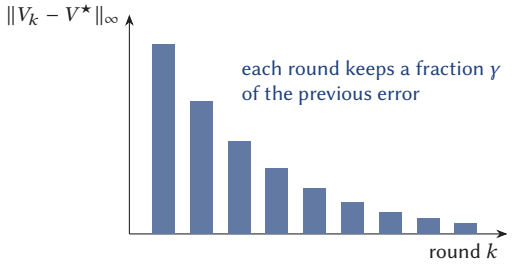
average with P, discount by γ, add r, take the max: the result is the new value of s

Value iteration, recalled (2/2): iterate until stable

Start from any table V_0 and back up *every* state at each round: $V_{k+1} = T^* V_k$.

- **Contraction**: one round brings any V a factor γ closer to $V^\star$, $\|T^*V - V^\star\|_\infty \leq \gamma \|V - V^\star\|_\infty$; after k rounds the initial error is multiplied by γ^k.
- **Performance loss** (Thm. 8.5): if V is within ε of $V^\star$ everywhere, its greedy policy loses at most $\frac{2\gamma}{1-\gamma}\varepsilon$.
- One round costs $|\mathcal{S}|$ backups: the size of the table.

The question of this lecture: what if V_{k+1} cannot be *stored*, only *fitted*?



Approximate value iteration: back up, then fit

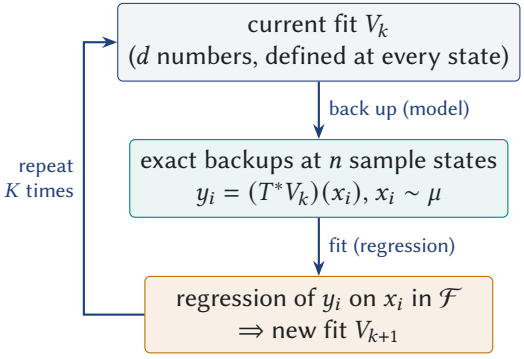
With a class $\mathcal{F}$ and a regression step $\Pi_{\mathcal{F}}$:

$$V_{k+1} = \Pi_{\mathcal{F}} T^* V_k.$$

Each round:

1. pick sample states $x_1, \dots, x_n$ from the design law μ;
2. compute the *exact* backup there, $y_i = (T^* V_k)(x_i)$ (the model is known, V_k can be evaluated anywhere);
3. regress the y_i on the x_i in $\mathcal{F}$: this is V_{k+1}.

The error of round k is what the fit missed, $\varepsilon_k = \|V_{k+1} - T^* V_k\|_\infty$: with a known model, only the **approximation** error of the class.



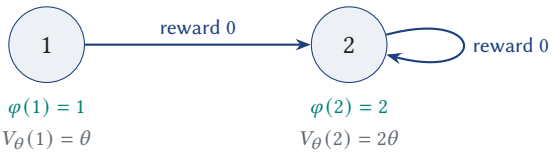
Output: the greedy policy of V_K,
read from the model by one more backup.

The toy example: two states and one straight line

Example 13.7, after Tsitsiklis and Van Roy (1996)

- **Two states** 1 and 2, **one action**: from 1 the system moves to 2, from 2 it stays.
- **All rewards are** 0, so the true value is $V^\star \equiv 0$. The model is known: the backup of any V is $(T^*V)(1) = \gamma V(2)$, $(T^*V)(2) = \gamma V(2)$.
- **The class**: one feature, $\varphi(1) = 1$, $\varphi(2) = 2$, and $V_\theta = \theta\varphi = (\theta, 2\theta)$. One number θ to fit; the values at the two states lie on a line through the origin. The class contains $V^\star$ ($\theta = 0$).

The simplest possible planning problem: exact backups, the answer in the class. Does back-up-then-fit find it?



true values: 0 and 0
fitted values: a line through the origin

A trace by hand

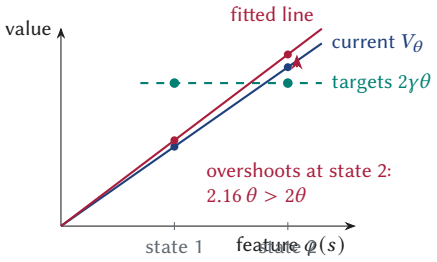
Example 13.7: one round of back-up-then-fit, with equal weights on the two states

Back up. Start from $V_\theta = (\theta, 2\theta)$. The backups are $(\gamma \cdot 2\theta, \gamma \cdot 2\theta)$: the *same* target $2\gamma\theta$ at both states.

Fit. Find the line through the origin closest to these two targets (least squares, equal weights):

$$\theta' = \tfrac{6}{5}\gamma\,\theta.$$

With $\gamma = 0.9$, θ is multiplied by 1.08 at every round: $1 \to 1.08 \to 1.17 \to 1.26 \to \cdots$ ($\times 2.2$ after ten rounds). Divergence as soon as $\gamma > \frac{5}{6}$, with exact backups and $V^\star$ in the class.



$\gamma = 0.9$, $\theta = 1$. A flat target is not a line through the origin: the compromise undershoots at state 1 and overshoots at state 2, where the value is largest.

What went wrong, and two ways out

The fit overshoots.
Both targets were $2\gamma\theta$, yet the fitted value at state 2 is $2.4\gamma\theta$: least squares does *not* keep the new values within the range of the targets. Backups shrink by γ, the fit expands by $\frac{6}{5}$, and $\frac{6}{5}\gamma > 1$ when $\gamma > \frac{5}{6}$.

Two lessons for the rest of Part 1: **where** you fit matters (the design law μ) and **how** you fit matters (averagers vs least squares). In practice least squares rarely diverges when the class is rich and μ broad.

Way out 1: fit where the system lives.
Put all the weight on state 2, the only state the chain visits in the long run: $\theta' = \gamma\theta$, and the iteration converges for every γ. It is the *only* weighting that converges for all γ (Exercise 13.3(c)).

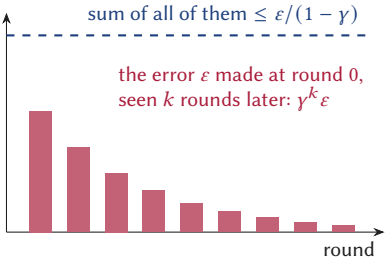
Way out 2: a fit that cannot overshoot.
Replace the line by a *cell average*: both states get the mean of the two targets, $2\gamma\theta$. The new values stay within the range of the targets; the iteration shrinks by γ and converges for every weighting (Exercise 13.3(d)).

Why errors do not explode: the $1/(1-\gamma)$ bill

Suppose every round makes a fitting error of at most ε.
The backup still contracts the past:

$$\|V_{k+1} - V^\star\|_\infty \;\le\; \gamma \, \|V_k - V^\star\|_\infty + \varepsilon.$$

- The error made at round k is multiplied by γ at every later round: it *fades*.
- Adding up all the fading errors: $\varepsilon(1 + \gamma + \gamma^2 + \cdots) = \varepsilon/(1-\gamma)$.
- $1/(1-\gamma)$ is the **effective horizon**: 10 for $\gamma = 0.9$, 100 for $\gamma = 0.99$.



A fitting error ε per round costs at most $\varepsilon/(1-\gamma)$ on the value, whatever the number of rounds. Contraction degrades *gracefully*.

The policy pays twice

First bill: along the rounds.
Fitting errors accumulate to $\varepsilon/(1-\gamma)$ on the value.

Second bill: value → policy.
A value within δ of $V^\star$ yields a greedy policy losing up to $\frac{2\gamma}{1-\gamma}\delta$: a wrong decision now, and its consequences over the horizon.

The same bill holds for *approximate policy iteration* (evaluate within ε, then improve; Prop. 13.10), plus one defect: the policies may cycle among near-optimal ones (*chattering*).

Theorem 13.4 (Error propagation in AVI, in short). If every round fits within ε (sup norm), then after K rounds

$$\|V^\star - V_K\|_\infty \le \frac{\varepsilon}{1-\gamma} + \gamma^K \|V^\star - V_0\|_\infty,$$

and the greedy policy π_K of V_K loses at most $\frac{2\gamma\,\varepsilon}{(1-\gamma)^2} + \frac{2\gamma^{K+1}}{1-\gamma}\|V^\star - V_0\|_\infty$.

With $\gamma = 0.9$, a fitting error of 1 is *allowed* to cost 180 on the policy. These are worst cases; the examples below show how far the truth is from them, and why.

A built-in alarm: the Bellman residual

We never know $V^\star$. But with the model we can always apply *one more backup* to a fit $\hat{V}$ and look at how much it moves: the residual $T^*\hat{V} - \hat{V}$.

Proposition 13.1 (Bellman residual bounds, in short). For every V, with π_V its greedy policy:
$\|V^\star - V\|_\infty \le \dfrac{\|T^*V - V\|_\infty}{1-\gamma}$ and $\|V^\star - V^{\pi_V}\|_\infty \le \dfrac{2\,\|T^*V - V\|_\infty}{1-\gamma}$.

- **Reading**: "if one backup barely moves V, then V is close to $V^\star$", with the usual horizon factor.
- It is a *certificate*: computable a posteriori, without knowing the answer. Pessimistic by about the factor $1/(1-\gamma)$, but it rings when the fit is wrong somewhere (the machine, below).
- Conversely a good V has a small residual: $\|T^*V - V\|_\infty \le (1+\gamma)\|V^\star - V\|_\infty$.

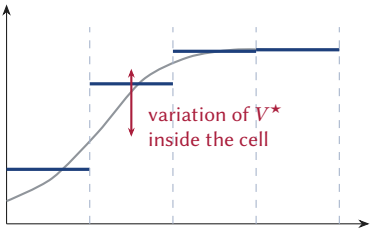
Why not minimize the residual directly, then? With the model one could; with samples its square is biased (Part 2).

Averagers: fits that never overshoot

Proposition 13.5 (Gordon, 1995), Corollary 13.6

An **averager** returns, at every state, a *weighted average* of target values (weights ≥ 0, sum ≤ 1): cell averages, k-nearest neighbors, interpolation, kernel smoothing.

- An average lies between the smallest and the largest of the numbers: the fit cannot leave the range of the targets. The backup contracts by γ, the fit does not expand: the loop contracts.
- So AVI **converges** to a fixed point $\hat{V}$, with $\|\hat{V} - V^\star\|_\infty \leq \|\Pi_{\mathcal{F}} V^\star - V^\star\|_\infty / (1-\gamma)$: best-approximation error, times the horizon.
- **State aggregation**: the best staircase is off by at most the *variation of $V^\star$ inside a cell*, so $\|\hat{V} - V^\star\|_\infty \leq \max_{\text{cells}} \operatorname{osc}(V^\star)/(1-\gamma)$: "as bad as the value varies inside a cell, times the horizon".



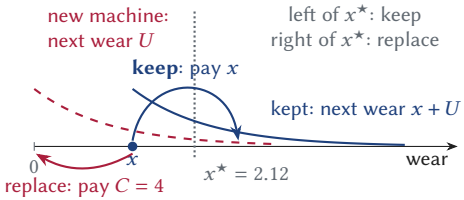
cell averages: a staircase that stays within the range of the curve

The worked example: machine replacement

Examples 8.15 and 13.8

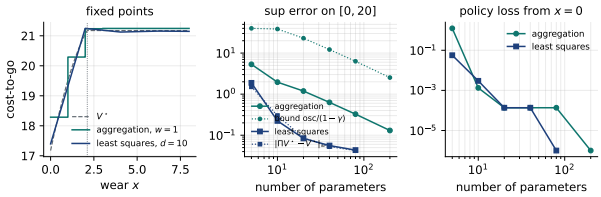
- **State**: the wear $x \geq 0$, a real number. **Actions**: *keep* or *replace*.
- **Dynamics**: kept, next year's wear is $x + U$; replaced, it is U (a new machine); U is exponential of mean 1.
- **Costs**: x per year while keeping, $C = 4$ per replacement; $\gamma = 0.9$; minimize the expected discounted cost.

Known answer: replace as soon as the wear exceeds $x^\star = 2.12$; cost 17.17 from a new machine. $V^\star$ rises from 17.17 to 21.17 on $[0, x^\star]$, then is *flat* (a machine about to be replaced costs the same whatever its wear).



The machine with cells of width 1

Example 13.8: approximate value iteration, exact backups, aggregation and least squares



The model is the grid of step 0.02 on $[0, 60]$ (3 001 states); every backup is exact; the exact solution is the reference. Design law: uniform on $[0, 20]$; cells of width w (left: $w = 1$) or d hat functions.

- **The fixed point is a staircase** (left, $w = 1$), with sup error 1.19; the best staircase approximation of $V^\star$ has error 1.21: the iteration adds nothing. Both errors sit in the cell of the threshold, where $V^\star$ bends; elsewhere the cell means are exact.
- **The bound is** 23.0: about 20 times too pessimistic; it pays twice for the one bad cell.
- **The greedy policy, read from the model**, is a threshold rule at 2.14 for $w \leq 1$, losing 10^{-4} (2.12 and 0 for $w = 0.1$). Only the coarsest grid, $w = 4$, is poor (threshold 3.26, loss 1.24).

Same machine, least squares on hat functions

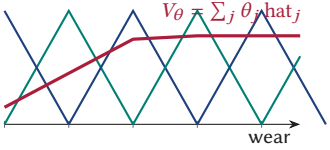
Example 13.8, continued

Now $\mathcal{F}$ = combinations of d hat functions, fitted by least squares under the same μ. No convergence theorem applies; it converges anyway.

- **Better per parameter**: sup error 0.22 for $d = 10$, 0.08 for $d = 20$, against 1.19 and 0.63 for as many cells; again about as good as the best fit of $V^\star$ in the class.
- **The policy is nearly optimal**: thresholds 2.08 ($d = 10$) and 2.14 ($d \geq 20$), losses $3 \cdot 10^{-3}$ and 10^{-4}.
- **The alarm**: certificates 12.5 ($w = 1$), 2.5 ($d = 10$), 0.9 ($d = 20$): ten times the true error, as $1/(1-\gamma) = 10$ predicts.

Value error 0.2 to 1; policy loss 10^{-4} to 10^{-3}. Why so different?

hat functions: one weight per knot, the fit is a broken line

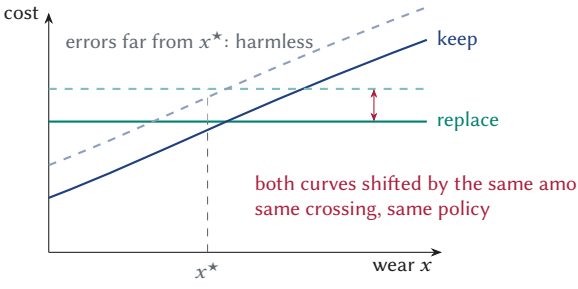


Why the policy is so much better than the value

At each wear x the decision compares two numbers, the cost of keeping and the cost of replacing: "replace iff keeping is dearer".

- Only the **sign of the difference** matters, and only near the threshold where the curves cross: far from it, errors of any size are harmless.
- An error that shifts both curves by the same amount does not move the crossing.
- The cost of a threshold rule is *stationary* at $x^\star$: a threshold error δ costs of order δ^2 (Example 13.11).

A value error is a number; a policy loss is a *difference of two numbers* at a few places. The bound $2\gamma\varepsilon/(1-\gamma)$ cannot see this.



Choosing where to fit: the design law μ

Example 13.8, least squares with $d = 10$ hat functions

With the model known, μ is *ours to choose*. Four choices on the machine:

design law μ	sup error on $[0, 20]$	alarm	greedy policy
uniform on $[0, 20]$	0.22	2.5	threshold 2.08, loss 0.003
uniform on $[0, 5]$	0.23	—	within 0.01 of the previous fit
occupation measure of $\pi^\star$	0.33	—	loss 0.006
uniform on $[5, 20]$: misses $[0, 5]$	off by 16 on $[0, 5]$	24	threshold 1.98, loss 0.03

- **Concentrating μ where the policy goes** changes little: hat functions are local.
- **Leaving out the region that matters** is another thing: the fit is off by 16 there and the alarm rings, 24 instead of 2.5. The *concentrability coefficient* of μ—how much rarer a state is under μ than under the policies we care about—is infinite. Even then the policy survives (loss 0.03).

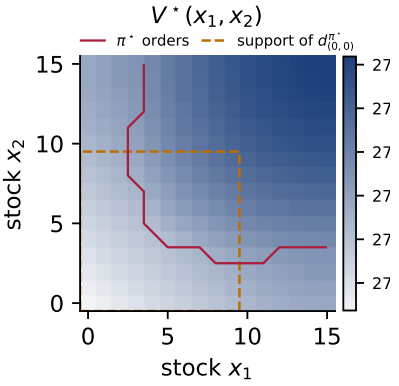
Fit where the policy goes, but keep a little mass everywhere; and read the alarm.

The second worked example: the bike store with two products

Example 13.9

- **State**: the stocks $(x_1, x_2) \in \{0, \dots, 15\}^2$, 256 states. **Action**: order-up-to levels (z_1, z_2).
- **Demands**: independent, geometric of mean 9 per week for each product.
- **Money**: price 1, purchase 0.5, storage 0.3 per bike per week, and one fixed cost 0.8 per delivery, whatever it contains (this couples the products).
- **Discount** $\gamma = 0.99$; **objective**: maximize the expected discounted profit.

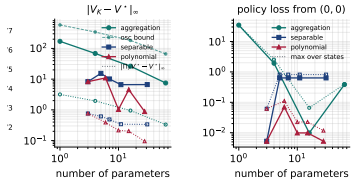
Known answer (solved exactly): $V^\star(0,0) = 271.4$, and $V^\star$ is flat to within 5.8. Optimal policy: a *can-order* rule—order as soon as one stock drops to 2 or below, up to 9.



$V^\star(x_1, x_2)$; orders below the crimson staircase; dashed square: where the optimal policy lives.

Two products: shape, level, and the policy

Example 13.9: AVI with exact backups, μ uniform; aggregation, separable $f_1(x_1) + f_2(x_2)$, polynomials



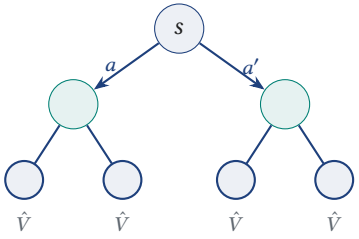
- **Aggregation gets the shape but loses the level.** With 16 cells the fixed point is 26.4 away from $V^\star$ (best staircase: 0.95). Each cell mean biases the backup slightly, and the bias is compounded over $1/(1-\gamma) = 100$ effective steps: 20 to 55 times the best approximation.
- **Yet the policies are excellent**: apart from the two coarsest aggregations, every greedy policy loses between 0.005 and 0.63 out of 271.4, less than a quarter of a percent, where Thm. 13.4 allows 1 800 to 55 000. The 16-cell policy, its value off by 26, loses 0.01: the decision compares backups at neighboring order levels, it reads *slopes*, not the level.
- **Not monotone in the number of parameters**: affine (3 parameters) loses 0.005, every richer separable class 0.63; 16 cells 0.01, 64 cells 0.39.
- **Design law**: under the occupation measure of $\pi^\star$, 26.4 $\longrightarrow$ 1.1 where it lives, but 156 states get no data; a 0.9/0.1 mixture with the uniform law: errors 1.4–2.0 everywhere, *cheap insurance*.

How good is my policy, really? Look ahead before trusting $\hat{V}$

Proposition 13.12, Example 13.13

With a model or simulator *at decision time*, do not play the greedy action of $\hat{V}$: solve the k-step problem from the current state with $\hat{V}$ as terminal value, and play its first action.

- **Guarantee**: a k-step lookahead on $\hat{V}$ is worth a value γ^{k-1} times more accurate (loss $\leq \frac{2\gamma^k}{1-\gamma}\|V^\star - \hat{V}\|_\infty$). Cost: a tree, independent of $|\mathcal{S}|$.
- **On the fitted machine values** (Part 2): the greedy policies cost 17.24 and 17.28 against 17.17. One step of lookahead: 17.178 and 17.181, losses divided by more than 15. Three steps: 17.1746, within 10^{-4} of the optimum, while the value error barely moves ($1.27 \longrightarrow 0.95$): the lookahead fixes the *difference* between the two actions.



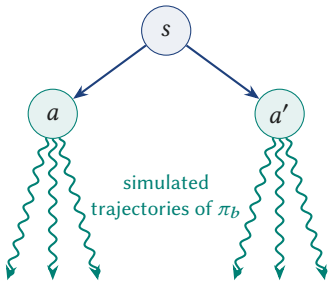
exact rewards and transitions for k steps, then trust $\hat{V}$ at the leaves

Rollout: simulate a base policy, then improve it

Exercise 13.5(a); Bertsekas et al. (1997) and Tesauro and Galperin (1997)

Take any reasonable **base policy** π_b. At the current state, for each action a: play a, then follow π_b in simulation, average the returns. Play the best action.

- This is the greedy policy of V^{π_b}, hence at least as good as π_b (policy improvement), computed *only at the states visited*, never as a table.
- **On the machine**: "never replace" costs 90 from a new machine; its rollout replaces as soon as $x > C(1-\gamma) = 0.4$ and costs 26.25; the optimum is 17.17.
- **Model predictive control**: re-solve an H-step problem at every step, apply the first action. **Monte Carlo tree search**: a lookahead tree grown selectively, rollouts at the leaves (Ch. 15).



Part 2

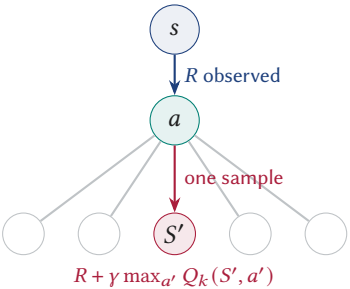
Without the model: fitted value iteration

1. Planning with a known model
2. Without the model: fitted value iteration
3. Policy evaluation with linear features: TD and its limits
4. When is approximation benign? Classes closed under the backup
5. Deep Q-networks

No model: the backup becomes a sample

Without P, the average over next states cannot be computed. With a **simulator** we get *one* successor S' and reward R for the pair (s, a), and form the target $Y = R + \gamma \max_{a'} Q_k(S', a')$.

- Y is a *noisy but unbiased* version of the backup: on average over S', the right number.
- A regression on many noisy targets recovers their mean: the fit of the Y_i **estimates** the fit of the exact backups.
- **Why Q rather than V?** With a model the action was read from V by one exact backup; without it, it must be read from the fit itself, $\arg\max_a Q(s, a)$.
- New error: **estimation**, of order $\sqrt{d/n}$ for a linear class.



the other successors are not seen: no model

The algorithm, line by line

Fitted Q-iteration (Ernst et al., 2005; Riedmiller, 2005); least-squares Monte Carlo (Longstaff and Schwartz, 2001)

Algorithm 13.1: Fitted Q-iteration (fitted value iteration in Q-form)

Input: simulator with a design law μ (regime (ii)) or batch of transitions (regime (iii)), class $\mathcal{F}$ with a regression oracle, iterations K, $Q_0 \in \mathcal{F}$

1 **for** $k = 0, \dots, K-1$ **do**

2 draw fresh transitions $(S_i, A_i, R_i, S_i')_{i \le n}$ with $(S_i, A_i) \sim \mu$ (simulator), or reuse the batch

3 $Y_i \leftarrow R_i + \gamma \max_{a'} Q_k(S_i', a')$ ▷ *noisy backups: the targets*

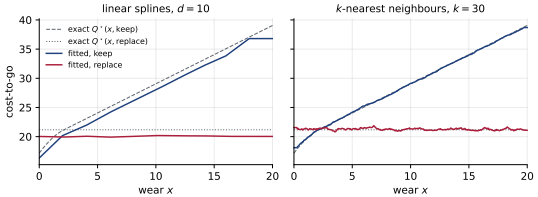
4 $Q_{k+1} \leftarrow \arg\min_{Q \in \mathcal{F}} \frac{1}{n} \sum_{i \le n} \big(Q(S_i, A_i) - Y_i\big)^2$ ▷ *regression*

Output: Q_K and its greedy policy $\pi_K(s) \in \arg\max_a Q_K(s, a)$

- Line 1 is the only line that knows the regime: **draw** (simulator, μ chosen) or **reuse** (batch, μ imposed). The rest is the back-up-then-fit loop of Part 1.
- Line 3: closed-form least squares for a linear class, a local average for k-NN, a training run for a network. To evaluate a fixed policy π, use the target $R_i + \gamma Q_k(S_i', \pi(S_i'))$.

Worked example: fitted Q-iteration for the machine

Example 13.11: regime (ii), the model used as a simulator



Same machine, 30 iterations. Left: $d = 10$ hat functions per action, $n = 10^4$ states drawn uniformly on $[0, 20]$ at each iteration, one sampled successor each. Right: 30-nearest neighbors on $n = 1\,000$ sample states.

- **Neither fit is accurate**: the linear one is off by up to 2.3 (at the right end, where its basis is constant while the cost of keeping grows) and puts the cost of a new machine at 16.3 instead of 17.2; the k-NN one is off by up to 0.9 and bumpy.
- **Yet the greedy policies** are threshold rules at 1.90 and 2.40, costing 17.24 and 17.28: less than 1% above the optimum. The bound $2\gamma\varepsilon/(1-\gamma) \approx 40$ is useless.
- **Against the exact backups of Part 1** (same basis, error 0.22): partly estimation error, partly the object fitted—$Q^\star(\cdot, \text{keep})$ grows linearly in the wear where $V^\star$ is flat (Exercise 13.9(e)).

From a fixed batch: the data decide

Regime (iii); Definition 13.3

Same algorithm, line 1 reads "reuse the batch". What changes is not the computation but **what can be learned**.

- μ is the law of the data. A pair the data never visit gets *no* target: its value is whatever the class extrapolates.
- **Coverage** is measured by the *concentrability coefficient* C: how much more often a pair is visited by the policies we care about than it appears in the data; infinite if the data miss a pair that matters. With a model, C was advice on choosing μ; here it is a fact about the data.
- **Offline reinforcement learning**: covering *every* policy is unrealistic; covering the *optimal* one ($C^\star$) is enough for a **pessimistic** learner that penalizes poorly covered pairs—the optimism of Ch. 15 run backwards (Rashidinejad et al., 2021; Kumar et al., 2020).
- The approximation error vanishes only if the class is *closed under the backup* (Part 4); containing $Q^\star$ is not enough (Chen and Jiang, 2019; Foster et al., 2022).

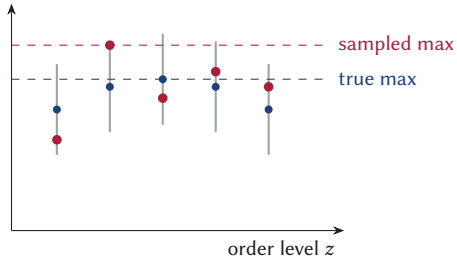
Regime (i): a little mass everywhere was cheap insurance. Regime (iii): the states the data miss are simply not learnable.

A caution: sampled backups are biased upward

Exercise 13.10(b), solved in Appendix F; Remark 12.21

The target takes a **maximum of noisy estimates**, one per action. The maximum of noisy numbers is, on average, larger than the maximum of their means: an upward bias at every round, compounded over the horizon.

- **Bike store, 16 cells**, the expectation replaced by the average of m sampled demands, 2 000 rounds: the signed error $V_K - V^\star$ is +171 for $m = 1$, +67 for $m = 10$, +35 for $m = 100$, against 26.4 with exact backups. Averaging the last 500 iterates does not help: a *bias*, not a fluctuation, decreasing like $1/\sqrt{m}$.
- **The policies hardly notice** (losses 0.31, 0.15, 0.15): the bias is nearly uniform, and the decision reads differences. Remedies: double Q-learning, double sampling.



blue: true values; red: one noisy estimate each; the largest red dot is usually too high

Related caution: do not minimize the squared sampled TD error

Part 3

Policy evaluation with linear features: TD and its limits

1. Planning with a known model

2. Without the model: fitted value iteration

3. Policy evaluation with linear features: TD and its limits

4. When is approximation benign? Classes closed under the backup

5. Deep Q-networks

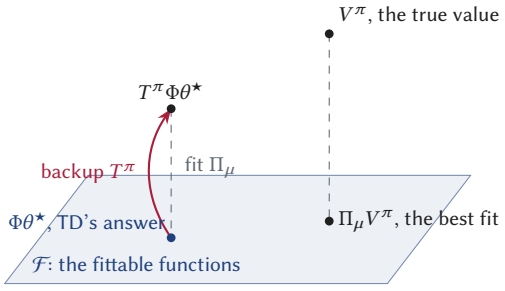
Evaluating a fixed policy: which equation does TD solve?

Linear class $V_\theta = \Phi\theta$, states sampled from μ, Π_μ the least-squares fit under μ.

- **Monte Carlo regression**: regress observed returns on the features. The answer is the *best fit* $\Pi_\mu V^\pi$: consistent, but needs complete episodes, and returns are very noisy.
- **Temporal differences** replace the return by $R + \gamma V_\theta(S')$: less noise, but the target depends on the current fit. The natural fixed point is

$$\Phi\theta = \Pi_\mu T^\pi \Phi\theta :$$

  the best fit of the backup of the fit, the *projected Bellman equation*.
- It is *not* the best fit $\Pi_\mu V^\pi$ in general.



What is guaranteed, and the two ways to compute it

Theorem 13.15 (Tsitsiklis and Van Roy, 1997); LSTD (Bradtke and Barto, 1996)

Theorem 13.15 (The projected Bellman equation, in words). If the states are sampled from the *stationary law of the evaluated policy*: (i) the projected equation has a unique solution $\theta^\star$, whose distance to V^π is at most $1/\sqrt{1-\gamma^2}$ times that of the best fit (in the μ-norm); (ii) linear TD(0) along a trajectory of the policy converges to it.

LSTD: solve a $d \times d$ system.
The equation reads $A\theta = b$, $A = \Phi^\top D(I - \gamma P^\pi)\Phi$, $b = \Phi^\top D r^\pi$. Estimate A and b by averages over the transitions and solve: every transition used once, no step size, cost $O(nd^2 + d^3)$. Session H05: $d = 10$ hat functions.

TD(0): the incremental version.
After each transition,
$\theta \leftarrow \theta + \alpha_t \big(R_t + \gamma \varphi(S_{t+1})^\top \theta - \varphi(S_t)^\top \theta\big)\varphi(S_t)$:
move the weights to reduce the TD error at S_t, the target held fixed ("semi-gradient").

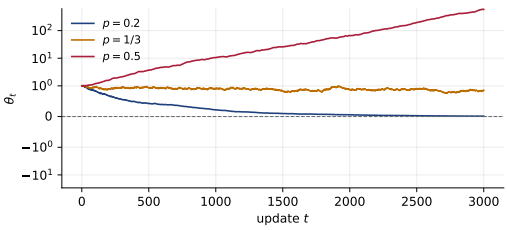
Why the stationary law? Under it, one step of the chain does not increase the μ-norm, so backup-then-fit still contracts. Under another law this can fail: next frame.

Off-policy TD can diverge

Example 13.16: the two states again

$\varphi(\mathrm{A}) = 1$, $\varphi(\mathrm{B}) = 2$; the policy moves A $\to$ B and stays; rewards 0, so $V^\pi \equiv 0$ and $\theta^\star = 0$ is *exactly* representable. TD updates from A with probability p, from B with probability $1 - p$ (stationary law: $p = 0$).

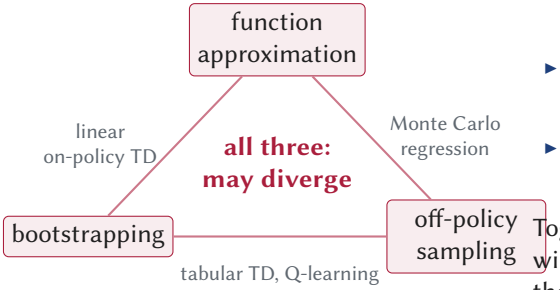
- From A: target $2\gamma\theta$, current value θ: pushed *up* if $\gamma > \frac{1}{2}$.
- From B: target $2\gamma\theta$, current value 2θ: pushed down.
- The up-pushes win iff $p > \frac{4(1-\gamma)}{3-2\gamma}$, i.e. $p > \frac{1}{3}$ for $\gamma = 0.9$.



Semi-gradient TD(0), $\gamma = 0.9$, constant step 0.01, three sampling laws.

The fixed point exists; the iteration is unstable. LSTD, which solves the equation instead of iterating, returns 0. Same two states as in Part 1: there the fit overshot; here the sampling law is wrong.

The deadly triad, in words



edges: any two of the three are safe in the settings of this book

Sutton and Barto (2018): three ingredients, each harmless alone.

- **Function approximation**: the fit moves values of states it was not asked about.
- **Bootstrapping**: the targets depend on the current fit.
- **Off-policy sampling**: states are not weighted as the evaluated policy visits them.

Together, the push from over-weighted states can win. "Any two are safe" describes our settings, not a theorem: with nonlinear fits even on-policy TD can diverge.

Repairs: reweight by the ratio stationary/sampling law; gradient-TD, a true gradient descent (Sutton et al., 2009); or avoid the iteration: LSTD, fitted Q-evaluation.

Part 4

When is approximation benign? Classes closed under the backup

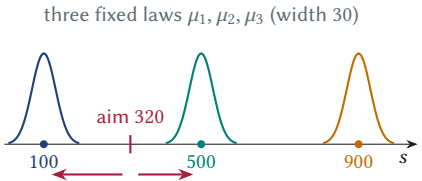
1. Planning with a known model
2. Without the model: fitted value iteration
3. Policy evaluation with linear features: TD and its limits
4. When is approximation benign? Classes closed under the backup
5. Deep Q-networks

Linear MDPs: "the next state is drawn from one of a few fixed laws"

Definition 13.17 (Jin et al., 2020); Example 13.19

A chain of 1000 positions; two actions aim at $s-50$ or $s+50$. Three **anchors** at 100, 500, 900, each with a fixed bell-shaped law μ_i around it. The next state comes from $P(s' \mid s,a) = \sum_{i=1}^{3} \phi_i(s,a)\,\mu_i(s')$, $\phi(s,a)$ being the *weights* of the intended position between its two nearest anchors; reward ϕ_2: aiming at the middle pays.

- A 2000×1000 transition matrix of **rank** 3: a *soft* state aggregation.
- **Everything is linear in** ϕ (Prop. 13.18): the backup of *any* V is $\langle \phi(s,a), w \rangle$ with $w_i = \vartheta_i + \gamma \sum_{s'} \mu_i(s') V(s')$: the backup of a fit is again a fit, and $Q^\star$ is in the class.



Planning inside a linear MDP: three numbers suffice

Example 13.19

Value iteration need only track $w \in \mathbb{R}^3$:

$$w_i \leftarrow \vartheta_i + \gamma \sum_{s'} \mu_i(s') \max_a \langle \phi(s', a), w \rangle,$$

which touches the 543 states in the supports of the three laws.

- **Cost**: about $5 \cdot 10^3$ operations per sweep against $2 \cdot 10^6$ for the tabular sweep, a ratio of 409, independent of the number of states.
- **Exactness**: after about 280 sweeps ($\gamma = 0.9$), the tabular $Q^\star$ and $\langle \phi, w^\star \rangle$ agree to $1.8 \cdot 10^{-13}$; $w^\star = (4.096, 7.699, 4.096)$: head for the middle.

But the structure is fragile. Mix a fraction η of a local random walk into the kernel: $Q^\star_\eta$ leaves the class. Keeping the features, the back-up-then-fit loop with exact backups and least squares

- converges, with error 0.023, 0.12, 0.25, 0.56 for $\eta = 0.01, 0.05, 0.1, 0.2$, against 0.007, 0.034, 0.068, 0.14 for the best fit of $Q^\star_\eta$: amplified by 3.4 to 4.1; same greedy rule;
- at $\eta = 1$ gives nonsense: $w = (8.6, 19.0, 8.6)$, above $V_{\max} = 10$—least squares overshooting again; interpolating at the anchors (an averager): error 3.3 instead of 9.6.

The error is a property of the pair (kernel, features), not of the features alone.

Bellman completeness: "the backup of a fit must be fittable"

Example 13.20: the classical complete class on a continuous state

Realizability: the answer $Q^\star$ belongs to the class. **Completeness**: the class is *closed* under the backup, $T^*\mathcal{F} \subseteq \mathcal{F}$—what the back-up-then-fit loop needs for the fit step to lose nothing. A linear MDP gives it for free; it can also hold without one:

The linear-quadratic regulator.
$S_{t+1} = S_t + A_t + U_{t+1}$ (noise of variance σ^2), reward $-q(s-\bar{s})^2 - a^2$. The *quadratics* $V_{p,c}(s) = -p(s-\bar{s})^2 - c$ (2 parameters) are closed under the backup: the best action is a linear feedback, the backed-up value again quadratic. Value iteration in the class is the *Riccati recursion* $p_k = q + \frac{\gamma p_{k-1}}{1+\gamma p_{k-1}}$; with $q = 1$, $\gamma = 0.9$, $\sigma = 1$: $p^\star = 1.588$, $c^\star = 14.29$, matching grid value iteration to 10^{-3}.

Not a linear MDP: Gaussian transition densities are not a finite mixture of fixed laws.

Fragile too: cost, dynamics and maximization conspire.

- Add a quartic term εe^4 to the cost: the backup of a quadratic is quartic, and enlarging the class to quartics does not help (the next best action is no longer linear).
- AVI with least squares under a Gaussian design law still converges (a Riccati recursion with $q + 6\varepsilon$): for $\varepsilon = 0.05$ the value is off by 1.6 on $|e| \le 3$, yet the feedback $a = -0.635\,e$ is within 0.05 of the optimal action and costs 0.01% more; for $\varepsilon = 0.2$ the value error is 6.1, the loss still 0.01%.

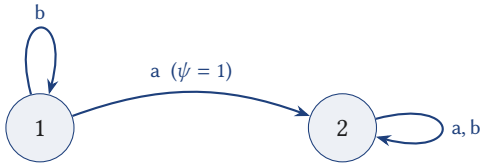
More features can break completeness

Example 13.21

Two states, two actions a, b, reward 1 on every transition. The pair $(1, \mathrm{b})$ leads back to state 1; every other pair leads to state 2.

- **Constants** $\{c\mathbf{1}\}$: the backup of c is $1 + \gamma c$, a constant. Complete.
- **Add one feature** $\psi = \mathbb{1}\{(s, a) = (1, \mathrm{a})\}$. Its backup is $1 + \gamma\, \mathbb{1}\{(s, a) = (1, \mathrm{b})\}$, since $(1, \mathrm{b})$ is the only pair whose successor offers $\psi = 1$. It varies at $(1, \mathrm{b})$, the class only at $(1, \mathrm{a})$. Not complete.
- $Q^\star \equiv \frac{1}{1-\gamma}$ is constant: *realizable* in both classes.

Realizability is about one function and improves with more features; completeness is about the whole class and may not (Weisz et al., 2021).



reward 1 everywhere

backup of ψ at $(1, \mathrm{b})$: $1 + \gamma \max_{a'} \psi(1, a') = 1 + \gamma$;
at every other pair: $1 + \gamma \max_{a'} \psi(2, a') = 1$.
Not in $\mathrm{span}\{\mathbf{1}, \psi\}$.

Part 5

Deep Q-networks

1. Planning with a known model
2. Without the model: fitted value iteration
3. Policy evaluation with linear features: TD and its limits
4. When is approximation benign? Classes closed under the backup
5. Deep Q-networks

Deep Q-networks (Mnih et al., 2015), line by line

49 Atari games from pixels, with a single architecture

Algorithm 13.2: Deep Q-learning with experience replay and a target network

Input: network Q_θ, replay memory $\mathcal{D}$, minibatch size B, target period C_{tgt}, ε, step η

initialize θ at random; $\bar{\theta} \leftarrow \theta$

for $t = 1, 2, \ldots$ **do**

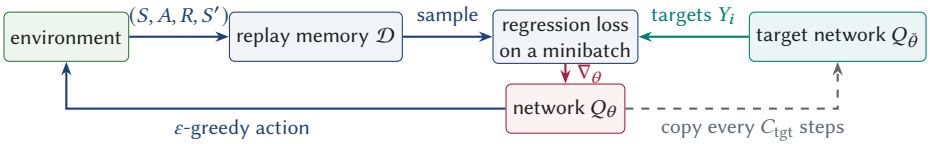
- choose A_t ε-greedily for $Q_\theta(S_t, \cdot)$; observe R_t, S_{t+1}; store (S_t, A_t, R_t, S_{t+1}) in $\mathcal{D}$ ▷ *act, log*
- sample $(S_i, A_i, R_i, S'_i)_{i \le B}$ uniformly from $\mathcal{D}$ ▷ *a minibatch of past transitions*
- $Y_i \leftarrow R_i + \gamma \max_{a'} Q_{\bar{\theta}}(S'_i, a')$ (R_i if terminal) ▷ *targets from the frozen network*
- $\theta \leftarrow \theta - \eta \nabla_\theta \frac{1}{B} \sum_{i \le B} \left(Q_\theta(S_i, A_i) - Y_i\right)^2$ ▷ *one regression step; Y_i treated as a constant*
- every C_{tgt} steps: $\bar{\theta} \leftarrow \theta$ ▷ *refresh the targets*

Regime (iv): the data come from the agent's own past behavior, the class is a network, the targets bootstrap. Read it as Algorithm 13.1 with the regression solved by gradient steps—next frame.

The two tricks, through dynamic-programming glasses

Target network = fitted iteration.
The targets are frozen for C_{tgt} steps: during a period the network solves *one* regression problem by C_{tgt} gradient steps. Each period is one round of fitted Q-iteration, and Thm. 13.4 says how the rounds' errors propagate. Without it the targets move at every step (Exercise 13.17).

Experience replay = the design law.
Minibatches from a memory of past transitions reuse each of them many times and decorrelate consecutive updates: the minibatch samples the mixture μ of recent state–action laws, which is what the regression oracle of Algorithm 13.1 assumes (Lin, 1992).



No escape from the deadly triad: off-policy data, nonlinear class, bootstrapped targets; no convergence theorem, instabilities are observed. Refinements fix identified defects: double DQN, dueling networks, prioritized replay, multi-step targets, distributional heads—Rainbow (Hessel et al., 2018).

Distributional heads: what changes, what does not

What changes: the output. Instead of one number $Q(s, a)$, a description of the *distribution* of the return:

- **C51** (Bellemare et al., 2017): a histogram on 51 fixed atoms, trained by cross-entropy after projecting the target onto the grid;
- **QR-DQN** (Dabney et al., 2018): N quantiles, trained by the pinball loss, no projection needed;
- **MaxEnt-DQN** (Bringas Miranda et al., 2026): m moments of a maximum-entropy law; the target moments are exact from one transition; $m = 8$ competes with 51 atoms.

The target is built from the frozen network at the action greedy for the *mean*.

What does not change.

- For evaluation, the distributional backup contracts like the ordinary one: the error propagation of Part 1 holds verbatim in Wasserstein distance (Exercise 13.16).
- For control, the guarantees still go through the means.
- The deadly triad and the coverage question do not move.

Distributional viewpoint

Why does it help deep agents? Not in tabular or linear settings (Lyle et al., 2019): with a network, the distribution is a richer training signal. What is gained is the ability to *read* risk from the output, not optimality for risk-sensitive criteria.

Summary

One loop, four regimes.
Back up, then fit. Known model: exact targets, choose μ, only the class can be wrong. Simulator: noisy targets, an estimation error. Batch: the data's coverage decides. Online: all of it along a trajectory.

Errors do not explode.
A fitting error ε per round costs at most $\varepsilon/(1-\gamma)$ on the value and $2\gamma\varepsilon/(1-\gamma)^2$ on the policy. Averagers always converge; least squares usually does, but can overshoot (two states: $\theta \mapsto \frac{6}{5}\gamma\theta$).

Value error ≠ policy loss.
Machine: value errors 0.2–2.3, policy losses 10^{-4} to 1%. Store: value off by 26, policy off by 0.01 out of 271. The decision reads differences and signs near the boundary; lookahead sharpens exactly that.

Where to fit, and with what.
Fit where the policy goes but keep a little mass everywhere; read the residual alarm. The class should be *closed under the backup* (linear MDPs, LQ), not merely contain the answer; more features can break this. TD diverges off-policy; DQN's tricks are the fitted loop and the design law, without a theorem.

To practice

Hands-on

H05 (`replacement.ipynb` and its continuation), the laboratory of this lecture:

- evaluate a fixed replacement policy by LSTD with $d = 10$ hat functions (try the Fourier basis);
- k-NN fitted Q-iteration ($n = 1\,000$, $k = 30$): check the threshold against $x^\star \approx 2.1$ and compute the costs on the reference grid;
- a network trained by TD, then deep Q-learning with targets from the *current* network: add a target network and a replay memory, and watch the training loss;
- find a better basis: $Q^\star$ is nearly affine in the wear except near the threshold.

Exercises (Chapter 13): 13.3 (the two-state example by hand: critical discount $\gamma_c(p)$, one-cell averager), 13.4 (approximate policy iteration, chattering), 13.9 (fitted Q-iteration for the machine; compare with the known-model loop), 13.10 (the store: a complete separable class; sampled backups and their bias), 13.11 (aggregation and polynomials on a 200-state riverswim, $\gamma = 0.997$), 13.12 (an American put: exact tree, regression with the model, least-squares Monte Carlo), 13.17 (target networks as fitted iterations), 13.18 (linear MDPs; completeness does not cure the triad).

References I

Agarwal, A. et al. (2026). *Reinforcement Learning: Theory and Algorithms*. Working draft of 27 June 2026.

Antos, A., C. Szepesvári, and R. Munos (2008). "Learning Near-Optimal Policies with Bellman-Residual Minimization Based Fitted Policy Iteration and a Single Sample Path". In: *Machine Learning* 71.1, pp. 89–129.

Bellemare, M. G., W. Dabney, and R. Munos (2017). "A Distributional Perspective on Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. PMLR, pp. 449–458.

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bertsekas, D. P., J. N. Tsitsiklis, and C. Wu (1997). "Rollout Algorithms for Combinatorial Optimization". In: *Journal of Heuristics* 3.3, pp. 245–262.

Bradtke, S. J. and A. G. Barto (1996). "Linear Least-Squares Algorithms for Temporal Difference Learning". In: *Machine Learning* 22.1–3, pp. 33–57.

Bringas Miranda, J., A. Garivier, and O. Cappé (2026). "A Novel Approach to Distributional Reinforcement Learning via Moment Matching". Under review; reference to be updated.

Chen, J. and N. Jiang (2019). "Information-Theoretic Considerations in Batch Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Vol. 97. PMLR.

Dabney, W., M. Rowland, M. G. Bellemare, and R. Munos (2018). "Distributional Reinforcement Learning with Quantile Regression". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

Ernst, D., P. Geurts, and L. Wehenkel (2005). "Tree-Based Batch Mode Reinforcement Learning". In: *Journal of Machine Learning Research* 6, pp. 503–556.

Foster, D. J., A. Krishnamurthy, D. Simchi-Levi, and Y. Xu (2022). "Offline Reinforcement Learning: Fundamental Barriers for Value Function Approximation". In: *Proceedings of the 35th Conference on Learning Theory (COLT)*. Vol. 178. PMLR.

References II

Gordon, G. J. (1995). "Stable Function Approximation in Dynamic Programming". In: *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pp. 261–268.

Hessel, M. et al. (2018). "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

Jin, C., Z. Yang, Z. Wang, and M. I. Jordan (2020). "Provably Efficient Reinforcement Learning with Linear Function Approximation". In: *Proceedings of the 33rd Conference on Learning Theory (COLT)*. Vol. 125. PMLR.

Kumar, A., A. Zhou, G. Tucker, and S. Levine (2020). "Conservative Q-Learning for Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems 33 (NeurIPS)*.

Le, H. M., C. Voloshin, and Y. Yue (2019). "Batch Policy Learning under Constraints". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Vol. 97. PMLR, pp. 3703–3712.

Lin, L.-J. (1992). "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8.3–4, pp. 293–321.

Longstaff, F. A. and E. S. Schwartz (2001). "Valuing American Options by Simulation: A Simple Least-Squares Approach". In: *The Review of Financial Studies* 14.1, pp. 113–147.

Lyle, C., M. G. Bellemare, and P. S. Castro (2019). "A Comparative Analysis of Expected and Distributional Reinforcement Learning". In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.

Mnih, V. et al. (2015). "Human-Level Control through Deep Reinforcement Learning". In: *Nature* 518.7540, pp. 529–533.

Munos, R. (2007). "Performance Bounds in L_p-Norm for Approximate Value Iteration". In: *SIAM Journal on Control and Optimization* 46.2, pp. 541–561.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. 2nd ed. Hoboken, NJ: Wiley.

References III

Rashidinejad, P. et al. (2021). "Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism". In: *Advances in Neural Information Processing Systems 34 (NeurIPS)*.

Riedmiller, M. (2005). "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method". In: *Machine Learning: ECML 2005*. Vol. 3720. Lecture Notes in Computer Science. Springer, pp. 317–328.

Scherrer, B. et al. (2015). "Approximate Modified Policy Iteration and its Application to the Game of Tetris". In: *Journal of Machine Learning Research* 16, pp. 1629–1676.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press.

Sutton, R. S. et al. (2009). "Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation". In: *Proceedings of the 26th International Conference on Machine Learning (ICML)*.

Tesauro, G. and G. R. Galperin (1997). "On-line Policy Improvement using Monte-Carlo Search". In: *Advances in Neural Information Processing Systems 9 (NIPS)*, pp. 1068–1074.

Tsitsiklis, J. N. and B. Van Roy (1996). "Feature-Based Methods for Large Scale Dynamic Programming". In: *Machine Learning* 22.1–3, pp. 59–94.

Tsitsiklis, J. N. and B. Van Roy (1997). "An Analysis of Temporal-Difference Learning with Function Approximation". In: *IEEE Transactions on Automatic Control* 42.5, pp. 674–690.

Weisz, G., P. Amortila, and C. Szepesvári (2021). "Exponential Lower Bounds for Planning in MDPs with Linearly-Realizable Optimal Action-Value Functions". In: *Proceedings of the 32nd International Conference on Algorithmic Learning Theory (ALT)*. Vol. 132. Proceedings of Machine Learning Research, pp. 1237–1264.